RISC Architecture
Ch 13

Some History

Instruction Usage Characteristics

Large Register Files

Register Allocation Optimization

RISC vs. CISC

Original Ideas Behind CISC

(CISC = Complex Instruction Set Computer)

- Make it easy target for compiler
  - small semantic gap between HLL source code and machine language representation
  - good at the time when compiler technology big problem
  - make it easier to design new, more complex languages
- Do things in HW, not in SW
  - addressing mode for 2D array reference?

Occam's Razor

(Occamin partaveitsi)

"Entia non sunt multiplicanda praeter necessitatem"
("Entities should not be multiplied more than necessary")
William Of Occam (1300-1349), English monk, philosopher

"It is vain to do with more that which can be done with less."

"I find that this really applies to ExtremeProgramming."
JosephPelrine

- The simple case is usually the most frequent and the easiest to optimise!
- Do simple, fast things in hardware and be sure the rest can be handled correctly in software

RISC Approach (2)

(RISC = Reduced Instruction Set Computer)

- Optimize for execution speed, instead of ease of compilation
  - compilers are good, let them do the hard work
  - compilers can be made even better (easily?)
  - do most important things very well in HW (e.g., 1-dim array reference or record reference) and the rest in SW (e.g., 3-dim. array references)
- What are *most important* things?
  - those that consume most of the time (in current systems?)
  - is this a moving target?

Amdahl's Law (5)

Speedup due to an enhancement is proportional to the fraction of the time (in the original system) that the enhancement can be used

Floating point instructions improved to run 2X; but only 10% of actual instructions are FP?

No speedup

$$ExTime_{new} = ExTime_{old} \times (0.9 * 1.0 + .1 * 0.5) = 0.95 \times ExTime_{old}$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{0.95} = 1.053$$

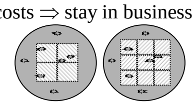
<< 2 !!!

Where is Time Spent? (6)

- Dynamic behaviour — Table 13.2 (Tbl 12.2 [Stal99])
  - execution time behaviour
- Which operations are most common?
- Which types of operands are most common? — Table 13.3 (Tbl 12.3 [Stal99])
- Which addressing modes are most common?
- Which cases are most common? — Table 13.4 (Tbl 12.4 [Stal99])
  - E.g., number of subroutine parameters?
- What is the case with current machines?

Original Ideas Behind RISC (3)

- Very large set of registers
  - more registers than can be addressed in any single machine instruction?
  - compilers can do good job in register allocation
- Very simple and small instruction set is faster
  - instruction pipeline is easy to optimise
- Economics
  - Simple to implement
    - ⇒ quickly to market ⇒ beat competition
    - ⇒ recover development costs ⇒ stay in business
  - Smaller chips are cheaper!



CISC Architecture (5)

- Large and complex instruction sets
  - direct implementation of HLL statements
    - case statement?
    - array or record reference?
- May be targeted to specific high level language — E.g., i432 and Ada
  - may not be so good for others — microJava, JEM?
- Many addressing modes
- Many data types

Vax11/780

char string, float, int, leading separate string, numeric string, packed decimal string, string, trailing numeric string, variable length bit field

Large Register File

Fig. 13.1 (Fig. 12.1 [Stal99])

- Overlapping register windows
  - fixed max nr (6?) of subroutine parameters
  - fixed max nr of local variables
  - function return values are directly accessible to calling routine in temporary registers
    - no copying needed
- I.e., when possible, use registers instead of stack for subroutine implementation

Problems with Large Register Files (2)

Fig. 13.2 (Fig. 12.2 [Stal99])

- What if run out of register sets?
  - save & restore values from memory (stack)
  - hopefully not very common
    - call stacks are usually not very deep!
    - find out from studies what is enough usually
- Global variables
  - store them always in memory?
  - use another, separate register file?

Register Files vs. Cache (2)

- Would it be better to use the same real estate (chip area) as cache?
  - register files have better locality — Table 13.5 (Tbl. 12.5 [Stal99])
  - caches are there anyway
  - caches solve global variable problem naturally
    - no compiler help needed
  - accessing register files is faster — Fig. 13.3 (Fig. 12.3 [Stal99])
- Third way to use the space for register files: register renaming
  - see next lecture on superscalar architecture

Register Allocation (3)

- Goal: Prob(operand in register) = high
- Symbolic register: any quantity that could be in register
- Allocate symbolic regs to real regs
  - if some symbolic regs are not used in same time intervals, then they can be assigned to the same real regs
  - use graph colouring problem to solve reg allocation problem

Graph Colouring Problem (2)

- Given a graph with connected nodes, assign n colours so that no neighbouring node has the same colour
  - topology
  - NP complete problem (see course on Design and Analysis of Algorithms)
- Application to register allocation
  - node = symbolic register
  - connecting line: simultaneous usage
  - no connecting line: can allocate symbolic registers to same physical register
  - n colors = n registers

Fig. 13.4

(Fig. 12.4 [Stal99])

How Many Registers Needed?

- Usually 32 enough (per register window!)
  - more ⇒ longer register address in instruction
  - more ⇒ no real gain in performance
- Less than 16?
  - Register allocation becomes difficult
  - not enough registers
    ⇒ store more symbolic registers in memory
    ⇒ slower execution

RISC Architecture (4)

- Complete one or more instructions each cycle (each instr. is still many cycles!)
  - read reg operands, do ALU, store reg result
  - all instructions are simple instructions
- Register to register operations
  - load-store architecture
- Simple addressing modes
  - easy to compute effective address
- Simple instruction formats
  - easy to load and parse instructions
  - fixed length

RISC vs. CISC

- Fixed instruction length (32 bits)
- Very few addressing modes
- No indirect addressing
- Load-store architecture
  - only load/store instructions access memory
- At most one operand in memory (load or store)
- Aligned data
- At least 32 addressable registers
- At least 16 FP registers

Table 13.8

(Tbl. 12.8 [Stal99])

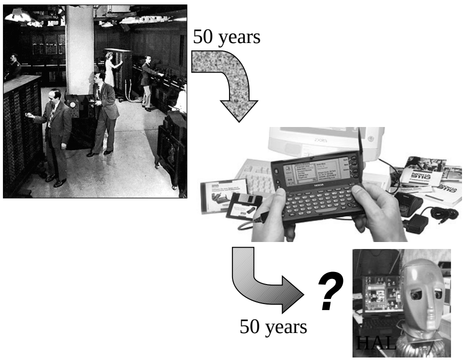
RISC & CISC United? (5)

- Pentium II, CISC architecture
- Each complex CISC instruction translated during execution (in CPU) into multiple fixed length 118 bit micro-operations (uop)
  - 1-4 uops/IA-32 (32 bit Intel Architecture) instruction
- Lower level implementation is RISC, working with RISC micro-ops
- Best of both worlds?
- Could CPU area/time be better spent without this translation?
  - Who wants to try? Transmeta Corporation?
  - Why? Why not?

RISC & CISC United? (3)

- Crusoe (by Transmeta) – emulate CISC
  - CISC architecture (IA-32, IA-64, Java?) visible to outside
- Each complex CISC instruction translated just before execution (in separate JIT translation with possibly optimized code generation) into multiple fixed length simple micro-operations
  - translation in SW, not in HW like with Pentium
- Lower level implementation is RISC, working with RISC micro-ops
  - VLIW (very long instruction word, 128 bits)
    - 4 uops/instruction (I.e., 4 *atoms/molecule*)

-- End of Chapter 13: History and RISC --



30.9.2002 37